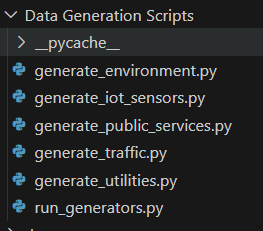
CitySim

**Step-1: Data Generation**

Simulated Data generation using python scripts. For creating realistic data, text files of lists of random states, zipcodes, cities are used. In the Folder data.



Components:

**1. Environment Data**

Captures real-time environmental conditions, including air quality, weather parameters, and alert notifications for a specific location.

**2. IoT Sensor Data**

Monitors building conditions such as occupancy, HVAC performance, energy efficiency, and environmental control in smart infrastructure.

**3. Public Services Data**

Logs emergency incidents such as fires, medical emergencies, and law enforcement responses, including severity, dispatched units, and casualties.

**4. Traffic Data**

Tracks real-time road conditions, congestion levels, accidents, weather impact, and traffic management measures to optimize urban mobility.

**5. Utilities Data**

Records energy consumption, water usage, waste management, and utility service disruptions for efficient resource monitoring in buildings.

**1.Environment Data Schema:**

| Field | Type | Description |
| --- | --- | --- |
| sensor\_id | string | Unique identifier for the environmental sensor |
| timestamp | string (ISO 8601) | Time of data recording |
| location | object | Location details |
| location.latitude | float | Latitude coordinate |
| location.longitude | float | Longitude coordinate |
| location.city | string | City name |
| location.state | string | State name |
| air\_quality | object | Air quality parameters |
| air\_quality.AQI | integer | Air Quality Index |
| air\_quality.PM2\_5 | integer | PM2.5 concentration |
| air\_quality.PM10 | integer | PM10 concentration |
| air\_quality.CO\_ppm | float | Carbon Monoxide level in ppm |
| air\_quality.NO2\_ppm | float | Nitrogen Dioxide level in ppm |
| air\_quality.VOC\_ppb | integer | Volatile Organic Compounds in ppb |
| air\_quality.SO2\_ppm | float | Sulfur Dioxide level in ppm |
| air\_quality.O3\_ppm | float | Ozone level in ppm |
| weather\_conditions | object | Weather-related conditions |
| weather\_conditions.temperature\_celsius | float | Temperature in Celsius |
| weather\_conditions.humidity\_percent | integer | Humidity percentage |
| weather\_conditions.wind\_speed\_kmh | float | Wind speed in km/h |
| weather\_conditions.precipitation\_mm | float | Precipitation in mm |
| weather\_conditions.ultraviolet\_index | integer | UV index |
| weather\_conditions.visibility\_km | float | Visibility in km |
| alerts | object | Alert information |
| alerts.smog\_alert | boolean | Indicates a smog alert |
| alerts.storm\_warning | boolean | Indicates a storm warning |
| alerts.heatwave\_alert | boolean | Indicates a heatwave alert |
| alerts.air\_quality\_alert | boolean | Indicates an air quality alert |

**2.IoT Sensor Data Schema**:

| Field | Type | Description |
| --- | --- | --- |
| sensor\_id | string | Unique identifier for the IoT sensor |
| timestamp | string (ISO 8601) | Time of data recording |
| building | object | Building details |
| building.name | string | Name of the building |
| building.building\_id | string | Unique building identifier |
| building.floor | integer | Floor number |
| building.room\_number | string | Room number |
| occupancy | object | Occupancy information |
| occupancy.people\_count | integer | Number of people present |
| occupancy.capacity\_limit | integer | Maximum occupancy capacity |
| occupancy.motion\_detected | boolean | Motion detection status |
| occupancy.entry\_exit\_logs | array | Logs of people entering/exiting |
| HVAC\_status | object | Heating, Ventilation, and Air Conditioning status |
| HVAC\_status.temperature\_setpoint\_celsius | integer | Target temperature in Celsius |
| HVAC\_status.current\_temperature\_celsius | float | Current temperature in Celsius |
| HVAC\_status.humidity\_percent | integer | Humidity level in percentage |
| HVAC\_status.system\_active | boolean | Whether HVAC system is active |
| HVAC\_status.air\_filter\_quality\_percent | integer | Air filter quality percentage |
| HVAC\_status.co2\_levels\_ppm | integer | CO2 levels in ppm |
| HVAC\_status.maintenance\_alerts | array | List of maintenance alerts |
| energy\_efficiency | object | Energy efficiency details |
| energy\_efficiency.power\_usage\_watts | integer | Power usage in watts |
| energy\_efficiency.smart\_lighting\_active | boolean | Smart lighting status |
| energy\_efficiency.energy\_savings\_mode | boolean | Whether energy-saving mode is enabled |
| reported\_by | string | Source reporting the data |

**3. Public Services Data Schema:**

| Field | Type | Description |
| --- | --- | --- |
| incident\_id | string | Unique identifier for the incident |
| timestamp | string (ISO 8601) | Time of incident recording |
| incident\_type | string | Type of public service incident |
| reported\_by | string | Source reporting the incident |
| location | object | Location details |
| location.building | string | Building name |
| location.latitude | float | Latitude coordinate |
| location.longitude | float | Longitude coordinate |
| severity | string | Severity level (Low, Medium, High) |
| emergency\_services | object | Emergency response details |
| emergency\_services.fire\_trucks\_dispatched | integer | Number of fire trucks dispatched |
| emergency\_services.ambulances\_dispatched | integer | Number of ambulances dispatched |
| emergency\_services.police\_units\_dispatched | integer | Number of police units dispatched |
| emergency\_services.drone\_surveillance\_used | boolean | Indicates if drone surveillance was used |
| response\_time\_minutes | integer | Response time in minutes |
| casualties | object | Casualty details |
| casualties.injured | integer | Number of people injured |
| casualties.fatalities | integer | Number of fatalities |
| damage\_assessment | object | Damage assessment details |
| damage\_assessment.estimated\_damage\_usd | integer | Estimated damage in USD |
| damage\_assessment.buildings\_affected | integer | Number of buildings affected |
| damage\_assessment.evacuations\_ordered | boolean | Indicates if evacuations were ordered |

**4. Traffic Data Schema:**

| Field | Type | Description |
| --- | --- | --- |
| event\_id | string | Unique identifier for the traffic event |
| timestamp | string (ISO 8601) | Time of traffic event recording |
| location | object | Location details |
| location.street | string | Street name |
| location.city | string | City name |
| location.state | string | State name |
| location.zip\_code | string | Zip code |
| location.latitude | float | Latitude coordinate |
| location.longitude | float | Longitude coordinate |
| traffic\_conditions | object | Traffic conditions data |
| traffic\_conditions.congestion\_level | string | Level of congestion (Low, Medium, High) |
| traffic\_conditions.average\_speed\_kmh | integer | Average speed in km/h |
| traffic\_conditions.vehicle\_count | integer | Number of vehicles detected |
| traffic\_conditions.accidents | array | Accident details |
| weather\_conditions | object | Weather conditions affecting traffic |
| traffic\_management | object | Traffic management strategies |

**5. Utilities Data Schema;**

| Field | Type | Description |
| --- | --- | --- |
| event\_id | string | Unique identifier for the utility event |
| timestamp | string (ISO 8601) | Time of utility event recording |
| building | object | Building details |
| building.name | string | Name of the building |
| building.building\_id | string | Unique building identifier |
| building.type | string | Type of building (Government, Commercial, Residential) |
| building.location.latitude | float | Latitude coordinate |
| building.location.longitude | float | Longitude coordinate |
| energy\_consumption | object | Energy consumption details |
| energy\_consumption.electricity\_kwh | integer | Electricity usage in kWh |
| energy\_consumption.gas\_m3 | integer | Gas usage in cubic meters |
| energy\_consumption.renewable\_energy\_percent | float | Percentage of renewable energy used |
| energy\_consumption.power\_outage\_reported | boolean | Indicates if a power outage was reported |
| water\_usage | object | Water usage details |
| waste\_collection | object | Waste collection details |

**Step-2: AWS Infrastructure - Terraform**

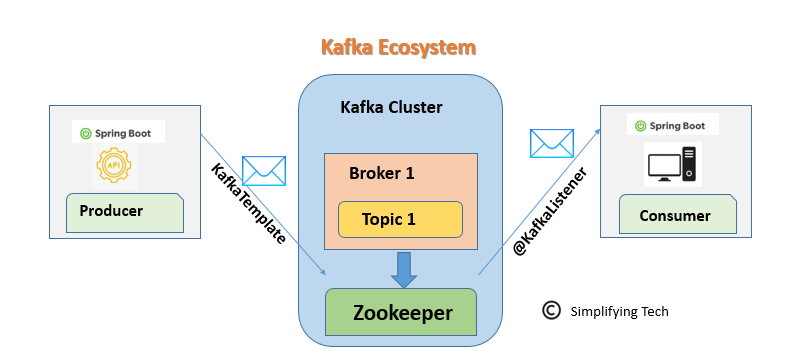
### ****Terraform Creates the Following AWS Resources:****

1. **S3 Buckets (Storage)**
   1. citysim-processed-data-bucket: Stores processed and cleaned data.
   2. citysim-raw-data-bucket: Stores raw, unprocessed data.
2. **CloudWatch Log Group (Monitoring)**
   1. /aws/citysim/logs: Logs system events and application activities.
3. **Security Group (Network Protection)**
   1. citysim\_sg: Controls inbound and outbound traffic.
   2. Allows traffic on **port 80 (HTTP)** and **port 443 (HTTPS)**.
   3. Allows **all outbound traffic**.
4. **Subnets (Networking)**
   1. subnet1 (10.0.1.0/24, Availability Zone: us-east-1a)
   2. subnet2 (10.0.2.0/24, Availability Zone: us-east-1b)
   3. subnet3 (10.0.3.0/24, Availability Zone: us-east-1c)
5. **VPC (Virtual Private Cloud)**
   1. citysim\_vpc (CIDR: 10.0.0.0/16)
   2. Enables **DNS resolution & hostnames** for AWS services.

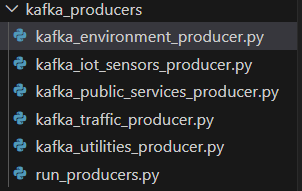
**Step-3: Apache Kafka Data Streaming**

Apache Kafka is a distributed event streaming platform designed for high-throughput, real-time data processing.

\*Make sure you download Apache Kafka, Spark and Hadoop.

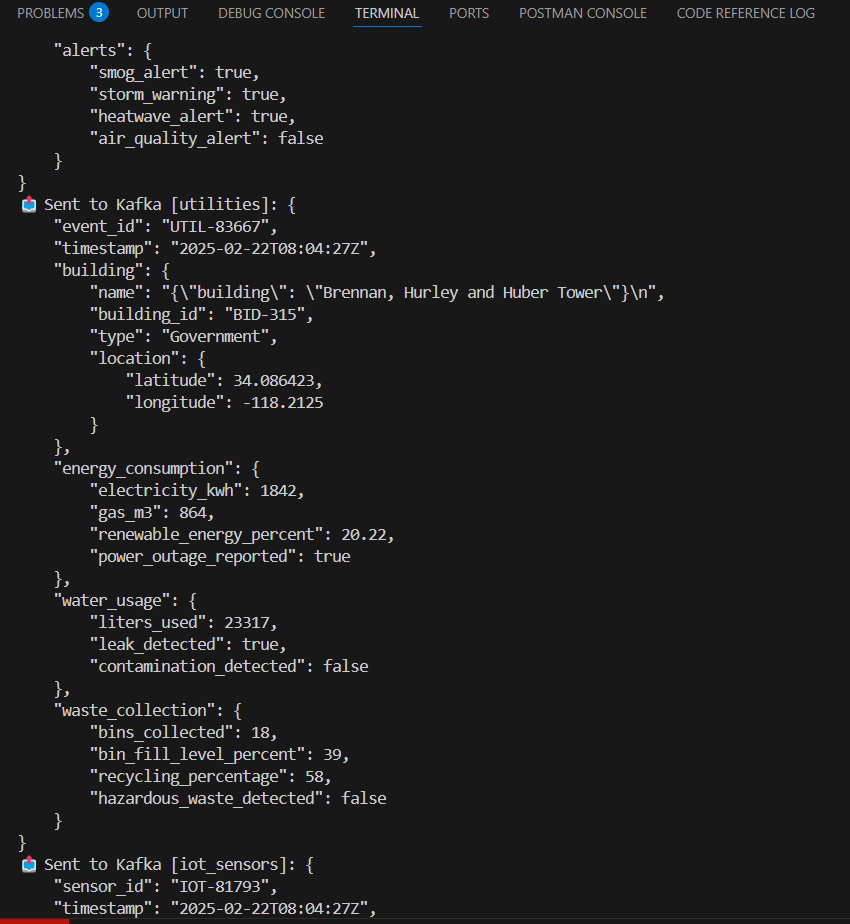


Spark Producers using generate data scripts generates data.



All Kafka producers are run using run\_producers.py. Kafka produces new data every 5 seconds.

**Output:**



**Step-4: Kafka Data Streaming and Apache Spark Data Processing**

Data produced by Kafka is sent to ZooKeeper through topics, is consumed by Kafka Consumer(spark\_kafka\_consumer.py)

### ****Usage of Apache Spark & Apache Kafka in the Pipeline****

#### ****1.Apache Kafka (Used for Data Ingestion & Streaming Source)****

* Kafka acts as the **message broker** that collects real-time data from various sources (Traffic, Utilities, Public Services, Environment, IoT Sensors).
* The Spark Streaming job **subscribes** to Kafka topics and **reads incoming data streams**.
* Kafka ensures **fault tolerance** and **high throughput** for data ingestion.

#### ****2.Apache Spark (Used for Streaming Processing & Analytics)****

* **Structured Streaming:** Processes real-time Kafka data using Spark **readStream**.
* **Schema Validation:** Parses JSON data and enforces structured schemas.
* **Time-based Aggregations:** Uses **watermarking & window functions** for 5-minute analytics.
* **Data Transformation:** Extracts, cleans, and restructures data for insights.
* **Storage Management:** Writes **raw data** to S3 and **aggregated insights** to a processed S3 bucket.
* **Live Monitoring:** Displays real-time analytics in the **console**.

Commands to start kafka server and to create topics:

**to start zookeeper**:

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

**to start server**:

.\bin\windows\kafka-server-start.bat .\config\server.properties

**To Create Topics**:

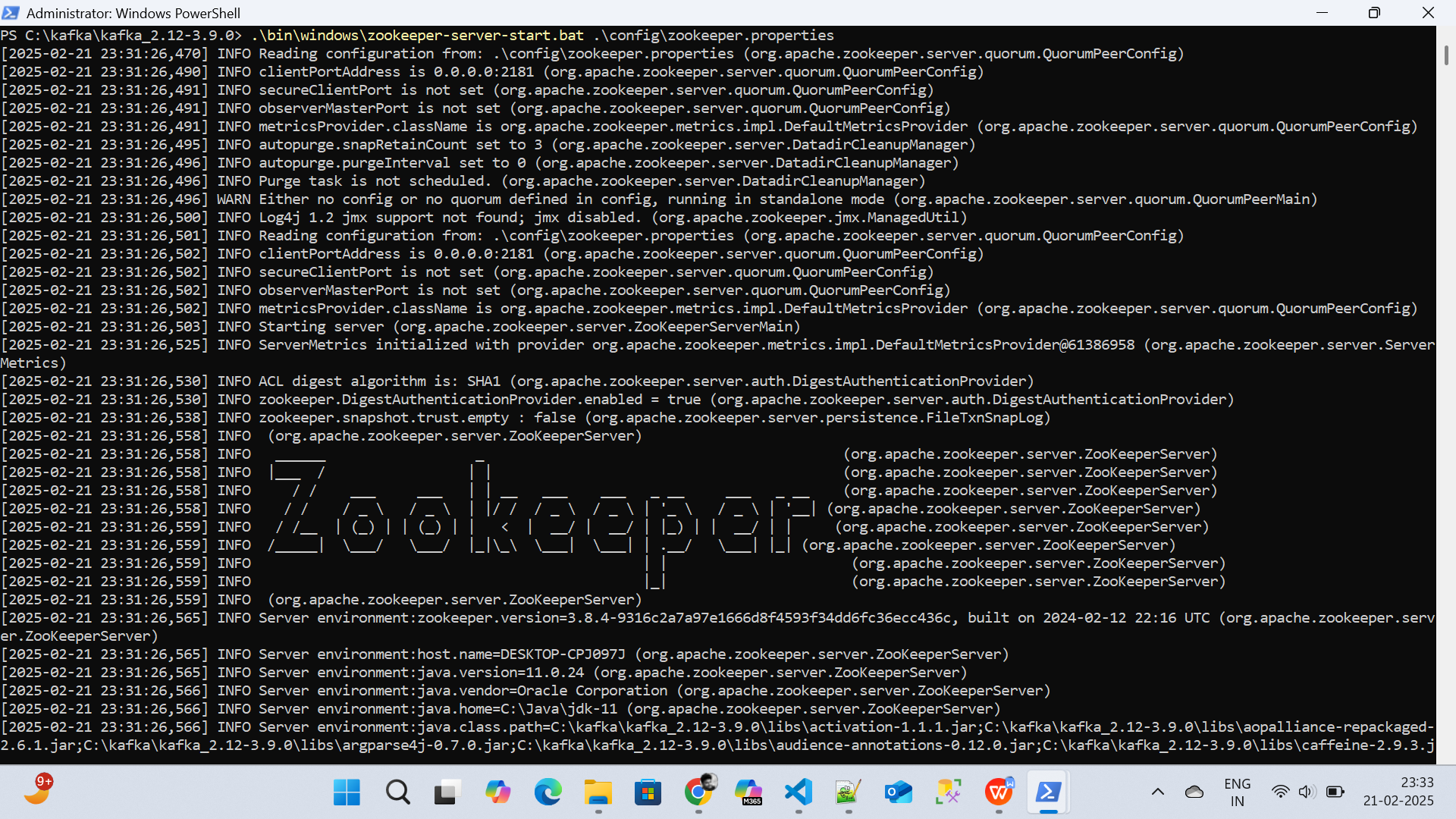
.\bin\windows\kafka-topics.bat --create --topic traffic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

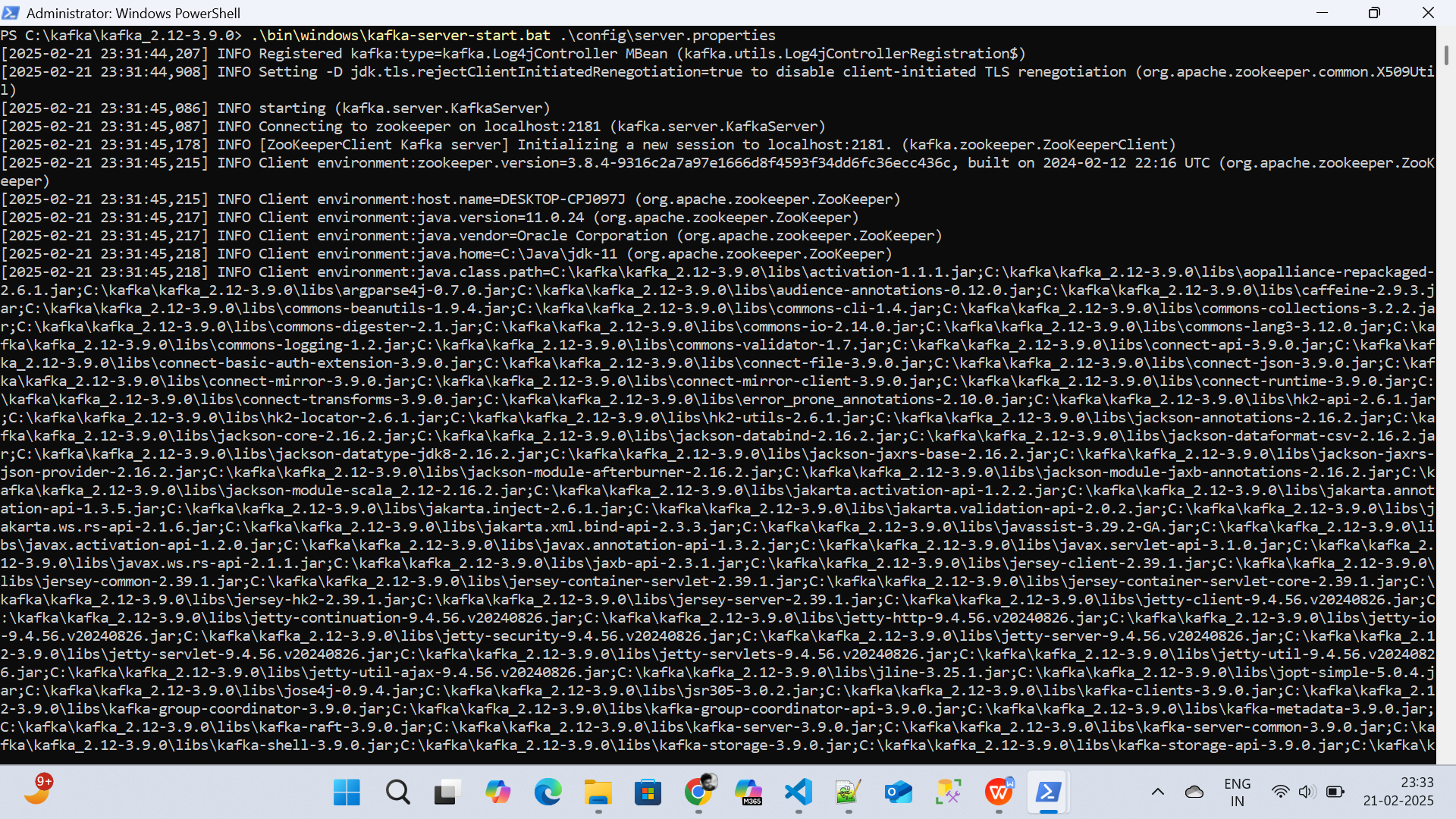
.\bin\windows\kafka-topics.bat --create --topic utilities --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

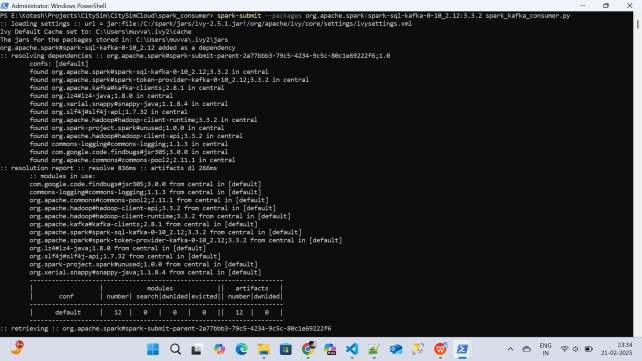
.\bin\windows\kafka-topics.bat --create --topic public\_services --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

.\bin\windows\kafka-topics.bat --create --topic environment --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

.\bin\windows\kafka-topics.bat --create --topic iot\_sensors --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

Output:  




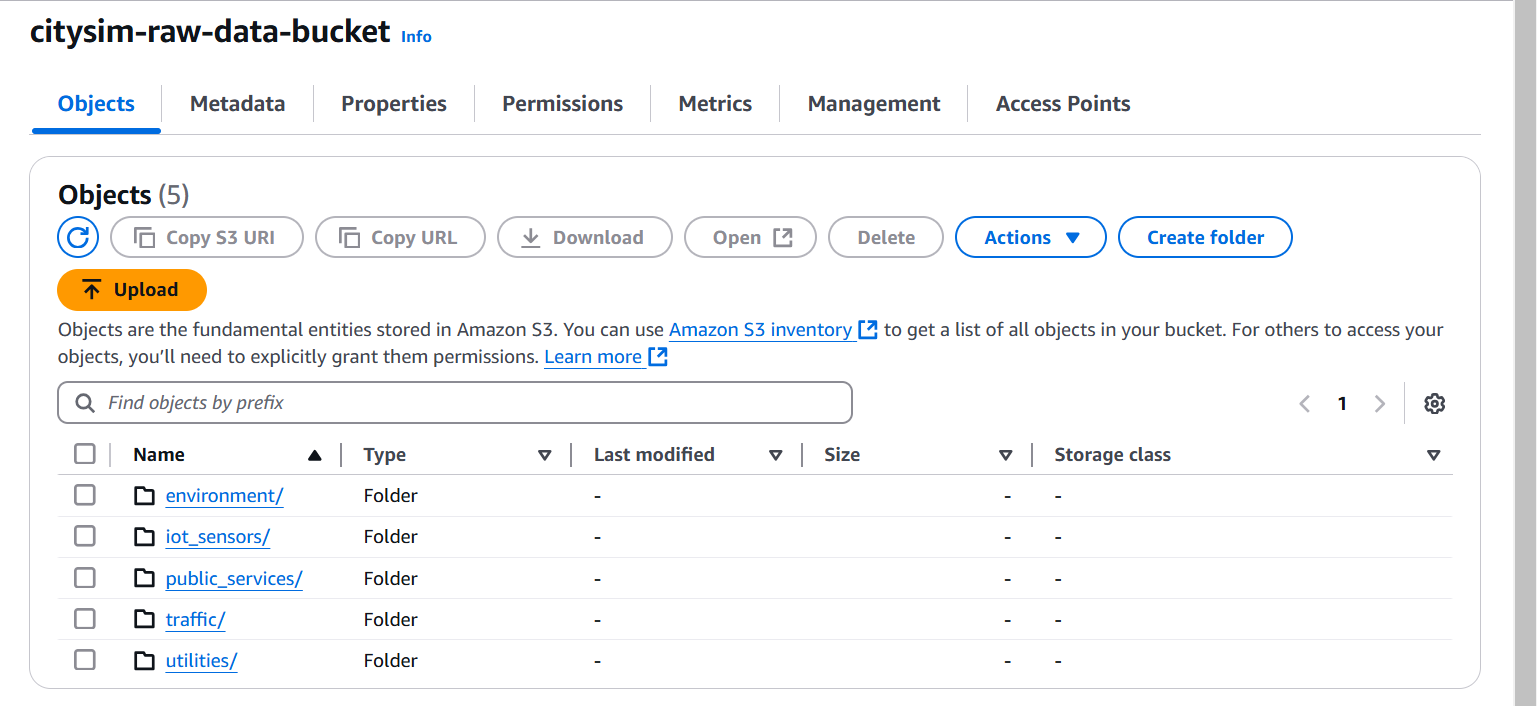


**Step-5: AWS S3**

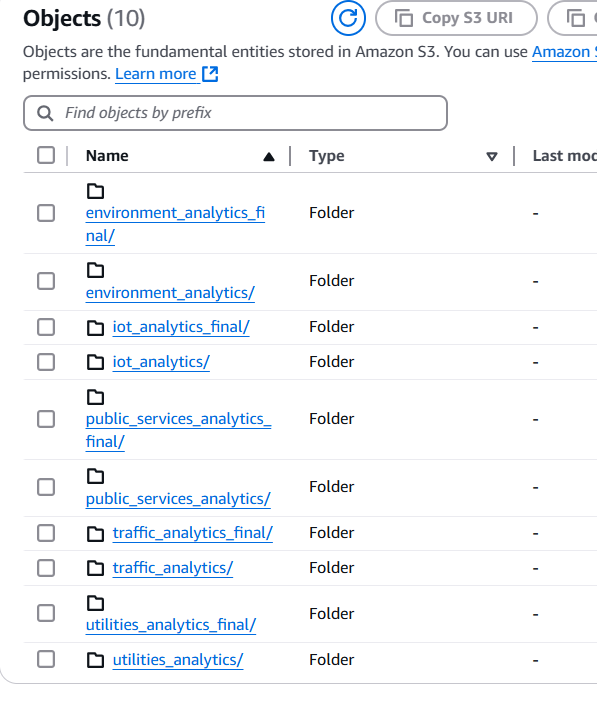
First, In aws s3, create five buckets,

1. citysim-etl-temp
2. citysim-glue-scripts
3. citysim-processed-data-bucket (to store aggregated results by spark)
4. citysim-raw-data-bucket (to store raw data consumed by Kafka)

**Raw bucket Screenshot:**



**Processed bucket Screenshot:**



**Step-6: Flattening the Processed data**

We have to flatten the latest Processed file data for every five minutes to send it to Dashboards through AWS OpenSearch.

**Step-7: Pushing to OpenSearch**

In OpenSearch Create the Domain, now run the AWS Glue Job to push flattened data from analytics\_final folder to OpenSearch.

The script pushS3ToOS.py contains the code to first delete previous indexes, and create new indexes in OpenSearch and pop them with data.

INDEX\_SCHEMAS = {

    "environment\_analytics": ["AQI", "count", "window\_start", "window\_end"],

    "iot\_analytics": ["avg\_people\_count", "building\_name", "window\_start", "window\_end"],

    "public\_services\_analytics": ["avg\_response\_time\_minutes", "incident\_type", "window\_start", "window\_end"],

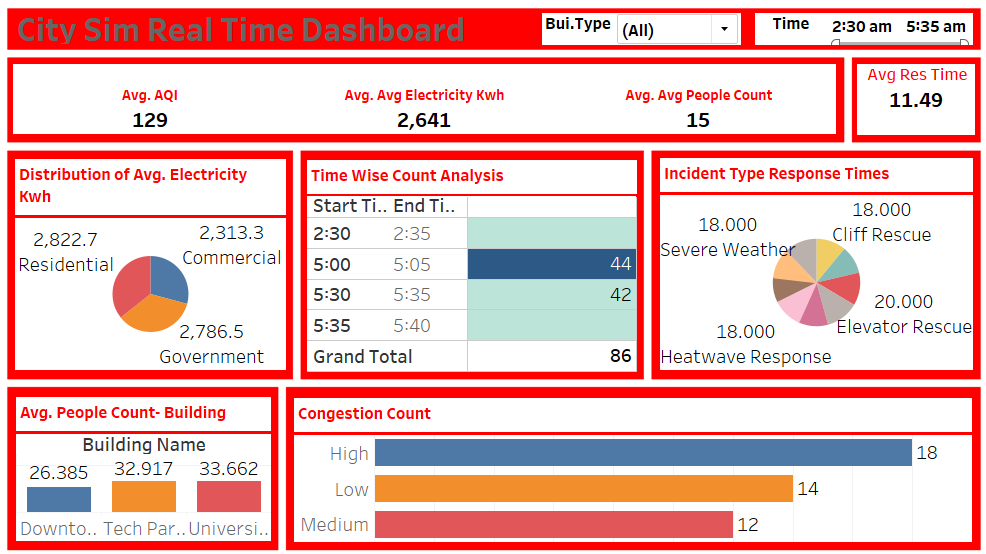
    "traffic\_analytics": ["congestion\_level", "count", "window\_start", "window\_end"],

    "utilities\_analytics": ["avg\_electricity\_kwh", "building\_type", "window\_start", "window\_end"]

}

**Step-8: Real Streaming Data Dashboards**

After popping data in OpenSearch. Create a python script to get data from formatted OpenSearch JSON Objects to CSV file, and connect that CSV file to Tableau with Live connection. Data gets refreshed immediately after changing.



**Step-9: Merging using AWS Glue Job**

The data generated in the last hour is merged using an AWS Glue Job. Glue Job is triggered every hour at 30 minutes And at 11:30 PM every night total cumulative files of whole day are generated, one whole raw and processed file is generated at 11:30 PM. The merged files are stored in respective raw and processed buckets.

merge\_hourly.py in the citysim-glue-scripts.

**Step-10: Flattening raw data using AWS Glue Job**

At 11:40 PM every night using Glue scheduled trigger, the raw data is flattened using script raw\_data\_flattener.py

Due to large size of raw data, batch processing is used. Output is written to

OUTPUT\_BUCKET = "s3://citysim-etl-data/"

TEMP\_BUCKET = "s3://citysim-etl-temp/"

Temporary bucket is used to store batch results.

Finally flattened csv file of raw data is created.

Raw data sample: {"topic":"environment","value":"{\"sensor\_id\": \"ENV-13830\", \"timestamp\": \"2025-02-22T05:31:21Z\", \"location\": {\"latitude\": 34.116455, \"longitude\": -118.244966, \"city\": \"Michelletown\\n\", \"state\": \"Missouri\\n\"}, \"air\_quality\": {\"AQI\": 145, \"PM2\_5\": 80, \"PM10\": 95, \"CO\_ppm\": 0.86, \"NO2\_ppm\": 0.044, \"VOC\_ppb\": 106, \"SO2\_ppm\": 0.143, \"O3\_ppm\": 0.168}, \"weather\_conditions\": {\"temperature\_celsius\": 30.41, \"humidity\_percent\": 68, \"wind\_speed\_kmh\": 11.43, \"precipitation\_mm\": 33.93, \"ultraviolet\_index\": 0, \"visibility\_km\": 7.51}, \"alerts\": {\"smog\_alert\": false, \"storm\_warning\": true, \"heatwave\_alert\": true, \"air\_quality\_alert\": false}}","ingestion\_time":"2025-02-21T21:05:00.071-08:00"}

After flattening:

| Attribute | Value |
| --- | --- |
| CO\_ppm | 0.86 |
| NO2\_ppm | 0.044 |
| O3\_ppm | 0.168 |
| PM10 | 95 |
| PM2\_5 | 80 |
| SO2\_ppm | 0.143 |
| VOC\_ppb | 106 |
| Air Quality | AQI: 145 |
| Air Quality Alert | FALSE |
| Alerts | smog\_alert: false |
| City | Michelletownn |
| Heatwave Alert | TRUE |
| Humidity (%) | 68 |
| Ingestion Time | 2025-02-21T21:05:00.071-08:00 |
| Location (Latitude) | 34.116455 |
| Longitude | -118.244966 |
| Precipitation (mm) | 33.93 |
| Sensor ID | ENV-13830 |
| State | Missourin |
| Storm Warning | TRUE |
| Timestamp | 2025-02-22T05:31:21Z |
| Topic | Environment |
| Ultraviolet Index | 0 |
| Visibility (km) | 7.51 |
| Weather Conditions | Temperature: 30.41°C |
| Wind Speed (km/h) | 11.43 |

Output is one row of flattened csv file.

**Step-11: Pushing Flattened raw data to RedShift Datawarehouse**

At 11:45 PM every Night, using Lambda function and EventBridge Rule,

Cron expression for EventBridge Rule, (LoadRedshiftDaily)

45 07 \* \* ? \*

Redshift Configuration:

1. Create New Worker Group, open query editor, and create tables with flattened raw data schema, with the lambda function, it runs copy command from flattened .csv to Redshift tables.

Redshift Schemas:  
Schema for file: utilities\_data

--------------------------------------------------

bin\_fill\_level\_percent BIGINT,

building VARCHAR(255),

building\_id VARCHAR(255),

contamination\_detected BOOLEAN,

energy\_consumption VARCHAR(255),

event\_id VARCHAR(255),

gas\_m3 BIGINT,

hazardous\_waste\_detected BOOLEAN,

ingestion\_time VARCHAR(255),

leak\_detected BOOLEAN,

location VARCHAR(255),

longitude DOUBLE PRECISION,

power\_outage\_reported BOOLEAN,

recycling\_percentage BIGINT,

renewable\_energy\_percent DOUBLE PRECISION,

timestamp VARCHAR(255),

topic VARCHAR(255),

type VARCHAR(255),

waste\_collection VARCHAR(255),

water\_usage VARCHAR(255)

Schema for file: traffic\_data

--------------------------------------------------

accidents VARCHAR(255),

alternate\_routes\_suggested VARCHAR(255),

average\_speed\_kmh BIGINT,

city VARCHAR(255),

event\_id VARCHAR(255),

fatalities DOUBLE PRECISION,

ingestion\_time VARCHAR(255),

injuries DOUBLE PRECISION,

latitude DOUBLE PRECISION,

location VARCHAR(255),

longitude DOUBLE PRECISION,

precipitation\_mm DOUBLE PRECISION,

road\_closure VARCHAR(255),

severity VARCHAR(255),

state VARCHAR(255),

timestamp VARCHAR(255),

topic VARCHAR(255),

traffic\_conditions VARCHAR(255),

traffic\_management VARCHAR(255),

vehicle\_count BIGINT,

vehicles\_involved DOUBLE PRECISION,

visibility\_km DOUBLE PRECISION,

weather\_conditions VARCHAR(255),

zip\_code VARCHAR(255)

Schema for file: public\_services\_data

--------------------------------------------------

ambulances\_dispatched BIGINT,

buildings\_affected BIGINT,

casualties VARCHAR(255),

damage\_assessment VARCHAR(255),

drone\_surveillance\_used BOOLEAN,

emergency\_services VARCHAR(255),

evacuations\_ordered BOOLEAN,

fatalities BIGINT,

incident\_id VARCHAR(255),

incident\_type VARCHAR(255),

ingestion\_time VARCHAR(255),

latitude DOUBLE PRECISION,

location VARCHAR(255),

longitude DOUBLE PRECISION,

police\_units\_dispatched BIGINT,

reported\_by VARCHAR(255),

response\_time\_minutes BIGINT,

severity VARCHAR(255),

timestamp VARCHAR(255),

topic VARCHAR(255)

Schema for file: iot\_sensor\_data

--------------------------------------------------

HVAC\_status VARCHAR(255),

air\_filter\_quality\_percent BIGINT,

building VARCHAR(255),

building\_id VARCHAR(255),

capacity\_limit BIGINT,

co2\_levels\_ppm BIGINT,

consumption\_trends VARCHAR(255),

current\_temperature\_celsius DOUBLE PRECISION,

energy\_efficiency VARCHAR(255),

energy\_savings\_mode BOOLEAN,

entry VARCHAR(255),

entry\_exit\_logs VARCHAR(255),

floor BIGINT,

hour VARCHAR(255),

humidity\_percent BIGINT,

ingestion\_time VARCHAR(255),

maintenance\_alerts VARCHAR(255),

motion\_detected BOOLEAN,

occupancy VARCHAR(255),

person\_id VARCHAR(255),

power\_usage VARCHAR(255),

reported\_by VARCHAR(255),

room\_number BIGINT,

sensor\_id VARCHAR(255),

smart\_lighting\_active BOOLEAN,

system\_active BOOLEAN,

timestamp VARCHAR(255),

topic VARCHAR(255)

Schema for file: environment\_data

--------------------------------------------------

CO\_ppm DOUBLE PRECISION,

NO2\_ppm DOUBLE PRECISION,

O3\_ppm DOUBLE PRECISION,

PM10 BIGINT,

PM2\_5 BIGINT,

SO2\_ppm DOUBLE PRECISION,

VOC\_ppb BIGINT,

air\_quality VARCHAR(255),

air\_quality\_alert BOOLEAN,

alerts VARCHAR(255),

city VARCHAR(255),

heatwave\_alert BOOLEAN,

humidity\_percent BIGINT,

ingestion\_time VARCHAR(255),

location VARCHAR(255),

longitude DOUBLE PRECISION,

precipitation\_mm DOUBLE PRECISION,

sensor\_id VARCHAR(255),

state VARCHAR(255),

storm\_warning BOOLEAN,

timestamp VARCHAR(255),

topic VARCHAR(255),

ultraviolet\_index BIGINT,

visibility\_km DOUBLE PRECISION,

weather\_conditions VARCHAR(255),

wind\_speed\_kmh DOUBLE PRECISION

**Step-12: Deleting All data at 12 PM.**

Using EventBridge Rule and Lambda function, deleting all buckets data, so that fresh data from the other day can begin. You can find all AWS scripts in folder.